API Gateway: пустая трата сил или полезный инструмент?

Василий Сошников

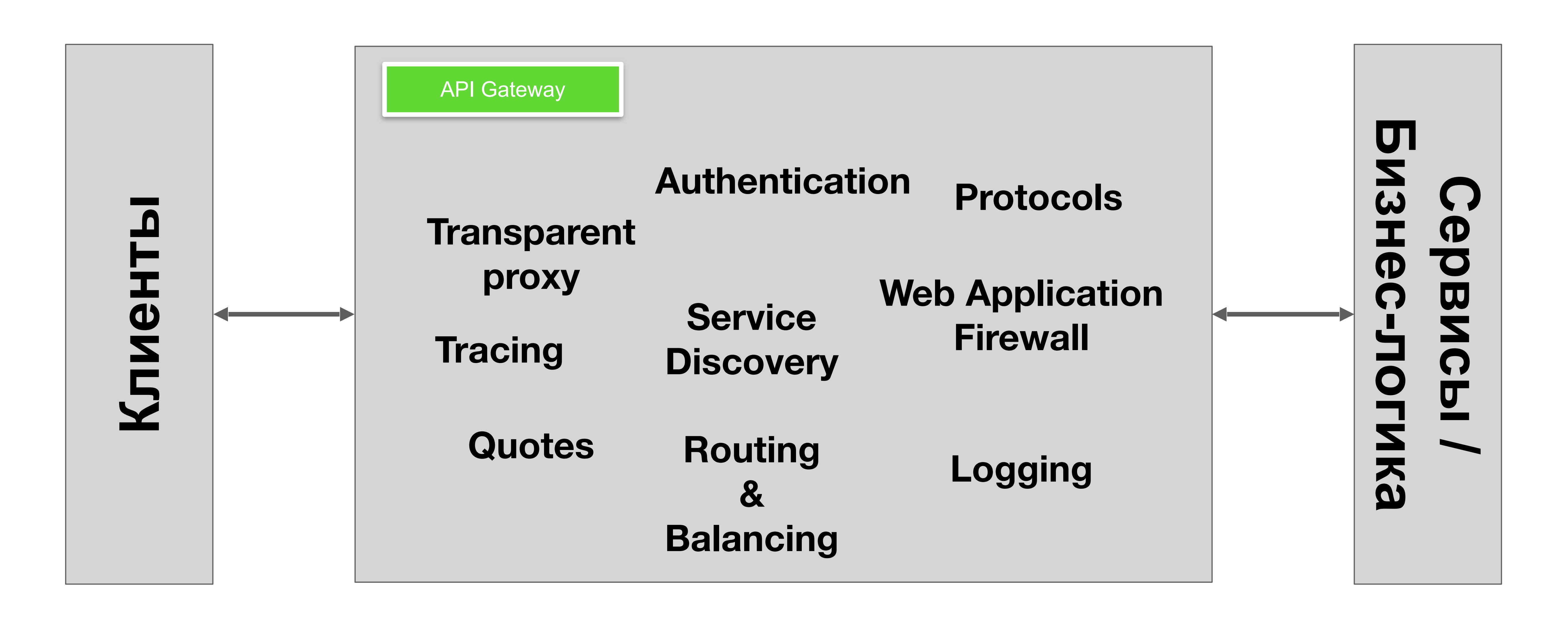


Содержание

- Разбор API Gateway на составляющие
- Попробую ответить на вопросы:
  Когда его нужно использовать AGW?
  Когда его не нужно использовать AGW?
- Немного о Service Mech, Service Discovery



Архитектура API Gateway



* *Главная точка входа любых запросов, все остальное API закрыто (речь не идет о системных интерфейсах, например, репликация).*

API Gateway

- API Gateway (AGW) — набор функций, подсистем, которые формируют AGW.

- Цель следующий секции — рассмотреть часто встречающиеся функции в AGW.



Authentication

- Единая точка для аутентификации пользователей и/или сервисов.
- Т.е. проверка всех входящих запросов на наличие доступов к ресурсам, если доступа нет, то разворачивать такие запросы в ***некий*** SSO.
- Другими словами — сервисам больше не нужно решать эту задачу.



Logging

- Все запросы пишутся в журнал, есть некий general logging — форматы стандартизированы.

- Данные журнала непрерывно пересылаются в некое хранилище, а то и несколько хранилищ, разных типов:

  - для аналитики, статистики;
  - для разбора проблем;
  - для мониторинга (например, соотношение 500’ок к 200’кам).

** Также это может быть основой для монетизации вашего API за счет: продажи аналитики, последующего биллинга и чарджинга API — а-ля CDR-логи. Если это “конечно” нужно.*



Logging / Пример

- Функция чуть сложней, попробую проиллюстрировать намеренно упрощенным примером из жизни:

Информация о запросах, ответах немного накапливается и отсылается в BDaS. Этот процесс ***никогда*** не влияет на взаимодействие с Bis services.

Доступ к накопленным данным.

Web, mobile, …

Архитектура Edge

NGINX

+ Custom log module

Bis services

BigData as service

Kafka

Buffer (ODS)

Hadoop

Casandra

Influx

Caches, services

Data API

Много кто потребляет

Некие бизнес-сервисы, которые имеют свой собственный жизненный цикл.

Buffer (часто встречал название ODS), она разделяется, накапливается, обогащается, разделяется.

Предвижу вопрос: почему не на Kafka? Ответ: не я проектировал.



Tracing

- Самое главное свойство: решение проблемы cycle requests (cycle breaker).

- Возможность наблюдения за пройденными путями запросов — крайне важное свойство в микросервисной архитектуре и mesh-oriented-архитектуре, также это важно в CDN, MDN.

** Плюс это все помогает анализировать проблемы, находить неоптимальные взаимодействия между разными элементами.*



Transparent proxy

- Опыт показывает, что это важная функция. И увы о ней часто забывают.
- Например HAProxy это умеет, NGINX нет, отчего:
- можно эмулировать.

  Системные способы: UDP/TCP Proxy Protocol v1, v2; HTTP X-Forwarted-For и т.п.;
  Не очень системные: добавлять данные (IP и т.п.) в некий payload своего протокола.

** ИБ, DevOps, SRE, DBA крайне уважают, если есть возможность понять, откуда (IP) приходит запрос.*

Жизненный пример: особенно драматичным может быть картина, при которой в мониторинге СУБД можно наблюдать только локальные IP клиентов (т.е. IP HAProxy).



Web Application Firewall

- Непрерывный поиск “опасных” запросов, ответов — т.е. WAF анализирует трафик и находит в нем угрозы безопасности.

- Фундаментальное отличие от “железных решений”: более быстрая эволюция решения, цена.

** См. также: https://www.cybintsolutions.com/cyber-security-facts-stats/, https://www.usatoday.com/story/money/2018/12/28/data-breaches-2018-billions-hit-growing-number-cyberattacks/2413411002/*



Quotes (квотирование запросов)

- Ограничение кол-во запросов к сервисам на основании разных политик, правил.

- Блокировка исполнения запросов. Хороший пример — вы планируете монетизировать свое API, то блокировка нужна.

- Позволяет избежать падений сервисов из-за чрезмерной нагрузки.

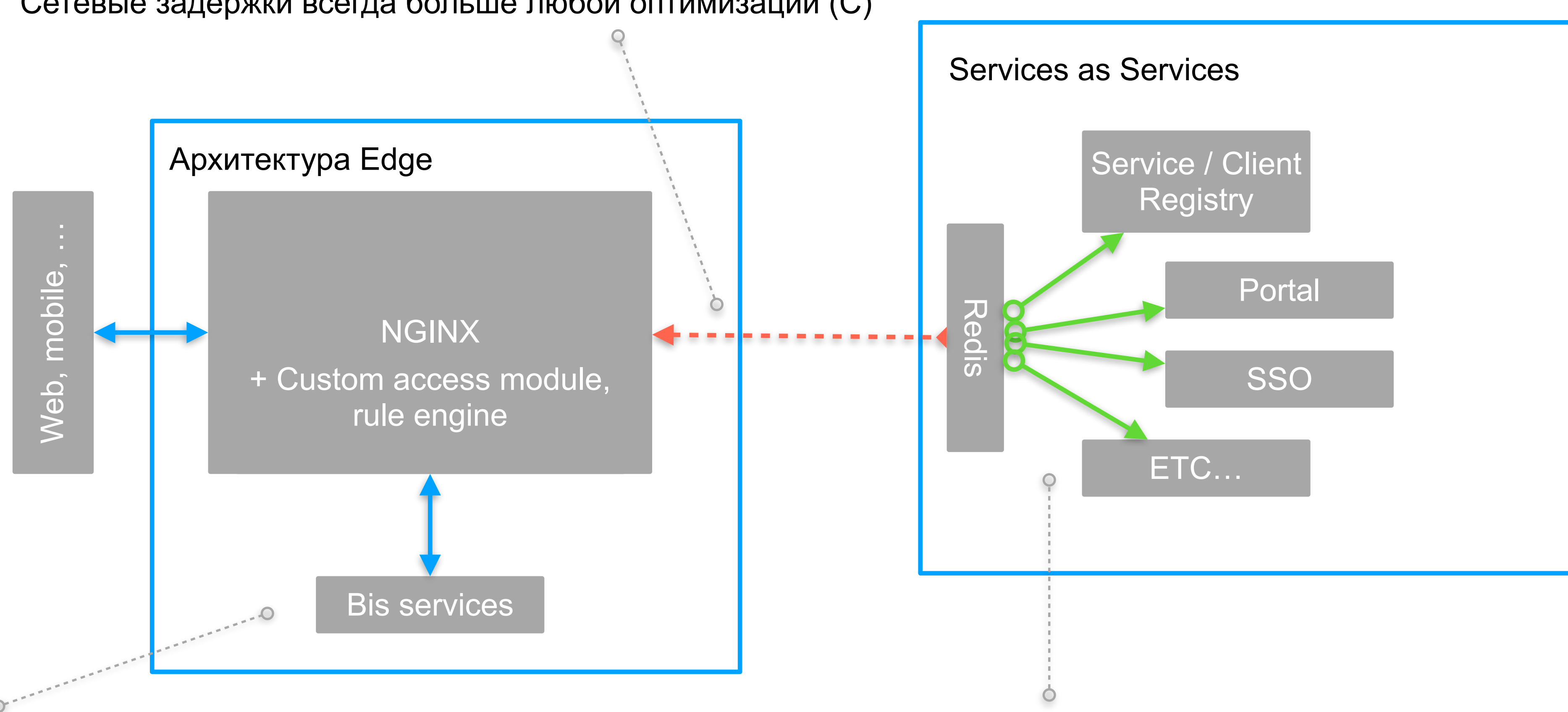
  Лучше отказать в обслуживании нескольким клиентам, чем наблюдать картину, когда упали все (С)



Quotes / Пример

- Функция неоднозначная, проиллюстрирую ***супер упрощённым*** примером из жизни:

В каком-то смысле данные о квотах на сервис, URL, etc реплицируются в модуль.
Предвижу вопрос: почему реплицируются?
Сетевые задержки всегда больше любой оптимизации (C)



Некие бизнес сервисы, которые имеют свой собственный жизненный цикл.

Тут действительно сложный процесс, и думаю что он в каждой компании свой.



Service Discovery

- Основная цель: контроль сервисов и инфраструктуры.

- Если вы решили использовать AGW, то ваша система *распределенная, большая и должна быть под контролем.

- Следовательно: лучше такую систему иметь, чем не иметь.
- Однако: охота отметить и процентную часть, и технологическую.



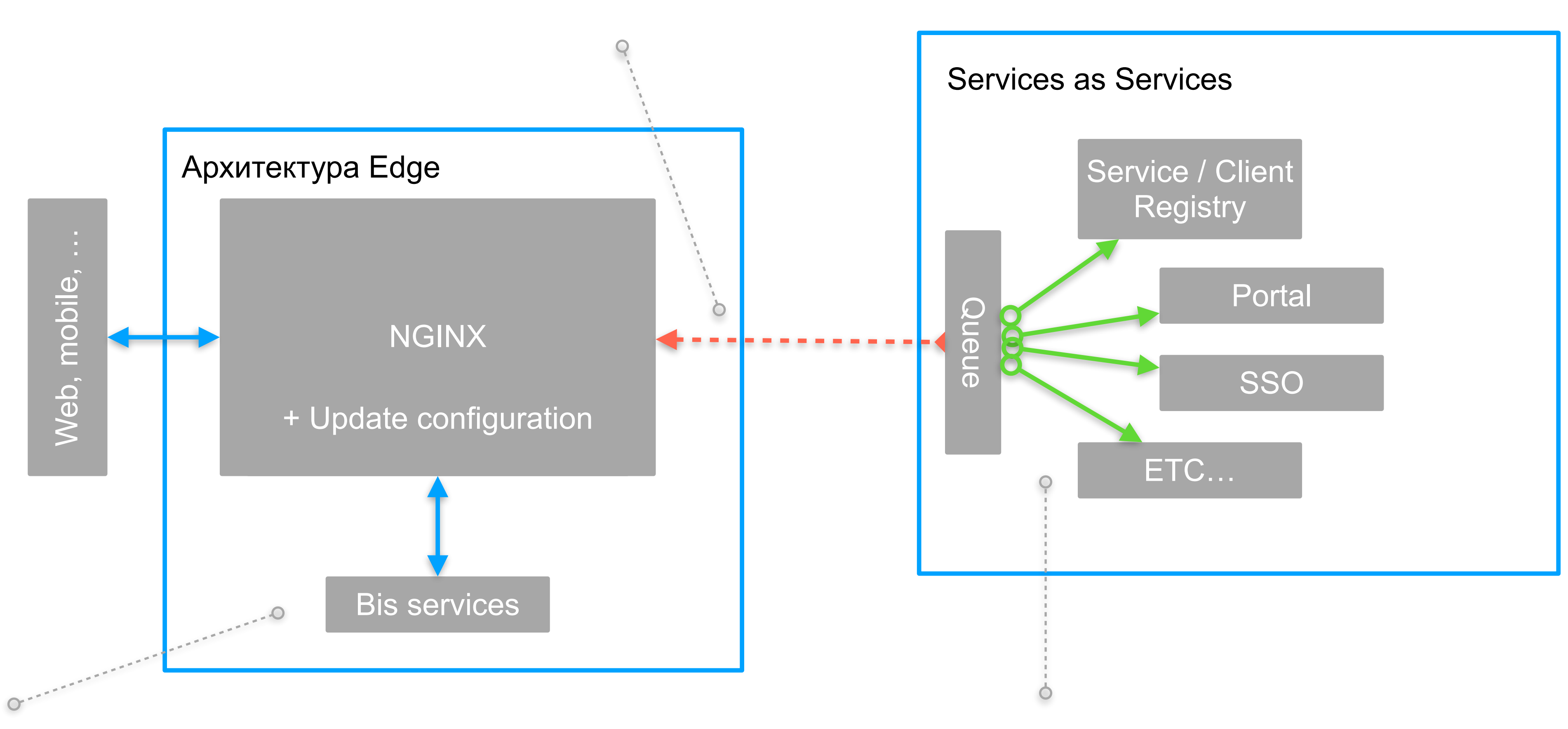
Service Discovery / Процессная часть

- Ключевые вопросы, из которых следует архитектура:
- Регистрация автоматическая или ручная?
- Связь этого процесса и CI/CD, тестирования?
- Как лучше связать эту систему с квотированием, конфигурированием? Нужно ли связывать? Например, хранить, обрабатывать все в некой одной системе?



Service Discovery / Архитектурная часть

Новая конфигурация распространяется методом push approach.



Некие бизнес-сервисы, которые имеют свой собственный жизненный цикл.
Сервис может быть вне Edge.

Тут действительно сложный процесс и думаю, что он в каждой компании свой.



Routing & Balancing

- Любая распределенная система должна знать о том, где и какие части в ней есть, какие точки входа (endpoints) есть, а каких нет.
- Направления, перенаправления, информация о точках входа — за это и отвечает routing.
- Хорошие примеры: Google, Amazon и их Routers.

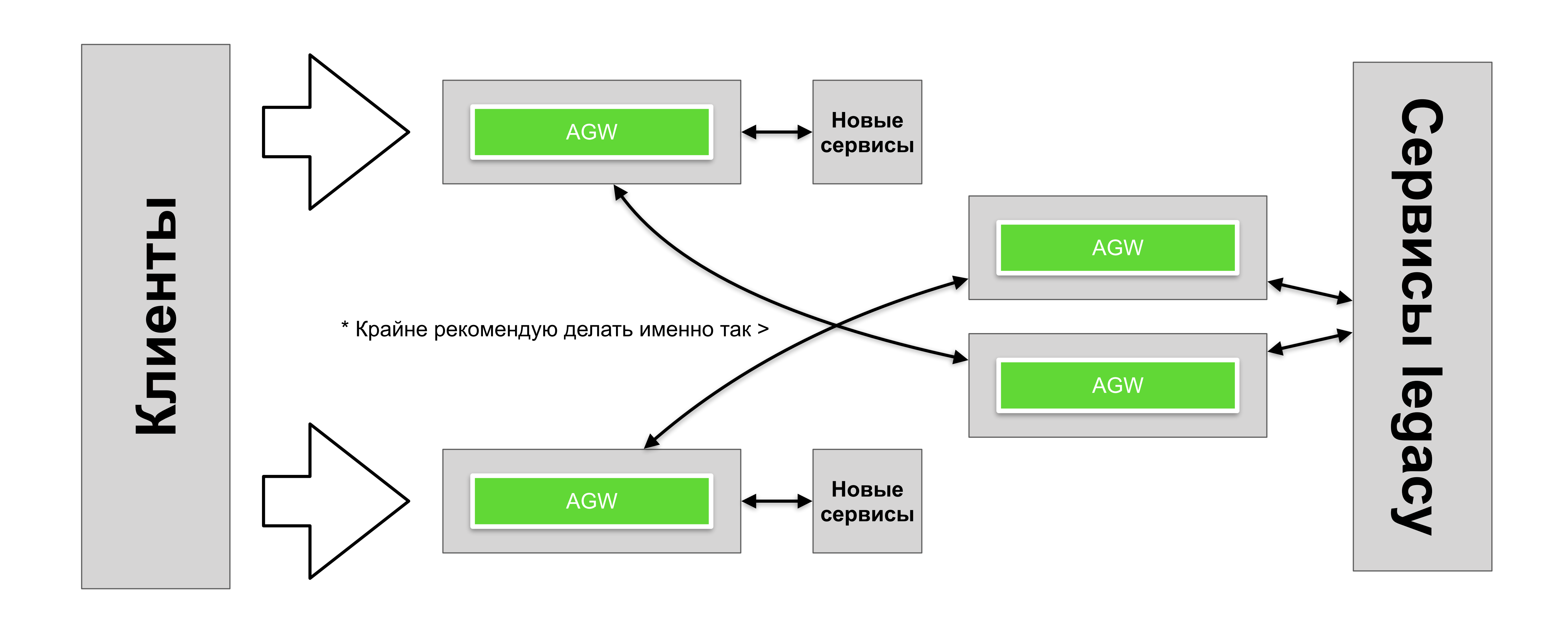


Protocols

- Кажется очевидным, но хочу отметить:
- ***Внешний мир***, как правило, требует что-то на входе HTTP(s) 1.1+, AMQP, MQTT, WebSockets (уже реже и реже).
- В то время, как ***внутренний мир*** может требовать: GRPC, JSON RPC, нечто иное на базе TCP, UDP.
- Отчего: если вы строите AGW, поддержка нескольких протоколов кажется необходимой.



Масштабируемость



- *AGW желательно должны уметь масштабироваться достаточно просто*
- *Правило в чуть более простом виде: AGW должна иметь доступность выше самого **доступного** сервиса, который он обслуживает, а скорость работы выше скорости работы самого быстрого из них.*

Промежуточный вывод

- Примечание:
  - Не все перечисленные функции обязательны
  - Обычно встречается некий набор функций, который реально нужен компании

- Pros:
  - Меньше разработки за счет того, что многие функции уже представляет AGW
  - Лучший контроль экосистемы, инфраструктуры, бизнес-сервисов

- Cons:
  - Решение сложное, добавить и внедрить новые функции в него — это сложный процесс
  - Сложно интегрировать уже в существующую экосистему.



Когда использовать?

- Если компания имеет и/или планирует иметь(!) большое кол-во проектов, и как след., проектных команд.
  Тогда AGW уменьшит *TCO и *TimeToMarket.

- Если компания планирует монетизировать API, тогда, ***но не всегда***, этому компоненту тоже быть.

* за счет генерализации функций, которые описаны выше



Когда не использовать?

- Один проект или несколько мелких проектов, и бурный рост не планируется

- Рассмотрим далее два диаметрально противоположных и радикальных кейса

** AGW гарантированно усложняет многое, но взамен дает вышеописанные свойства. Соответственно, вопрос заключается: вам нужны они все, или только какие-то из?*



(Анти)пример #1: Мелкий проект

- Единица измерения, которую я рекомендую использовать, — время
- 5 Web/Mobile -проектов
- Минимальный набор функций в современных проектах: Logging, Auth, Routing/Balancing
- Предположим, что на их реализацию каждая из 5 команд израсходовала 3 человеко-недели
- 15 человеко-недель или 3 месяца. Без учета поддержки своих реализаций.



Пример #2:
Крупный проект

- 20 Web/Mobile-проектов, 10 middleware (связь между core и Web/Molile-сервисами), при этом постоянно появляются новые, удаляются старые

- Минимальный набор функций в современных проектах:
  Logging, Auth, Routing/Balancing (***его уже недостаточно для такого кол-ва проектов***)

- Используя тот же принцип, получаем: 22 человеко-месяца, почти 2 года. Без учета поддержки своих реализаций.



Добавим к выводу

- AGW нужно использовать и внедрять тогда и только тогда, когда стоимость внедрения и поддержки будет меньше создания ограниченого кол-ва функций в рамках проектов



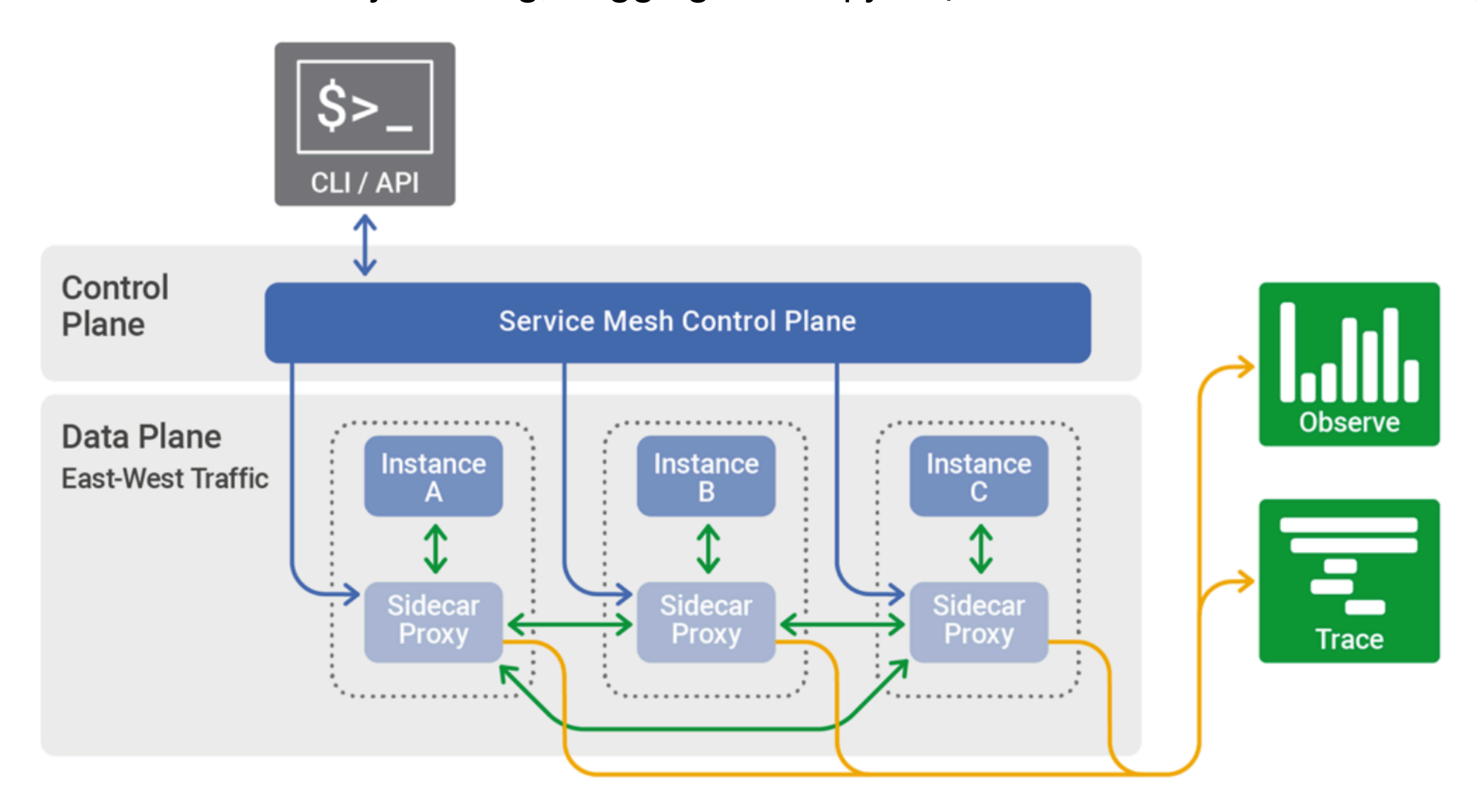
Примечание:
AGW и “процессы”

- Внедрение AGW — стандартизация ряда процессов, к этому надо быть готовым
- Команды надо обучить, они должны уметь этим всем пользоваться



Примечание: Service Mesh

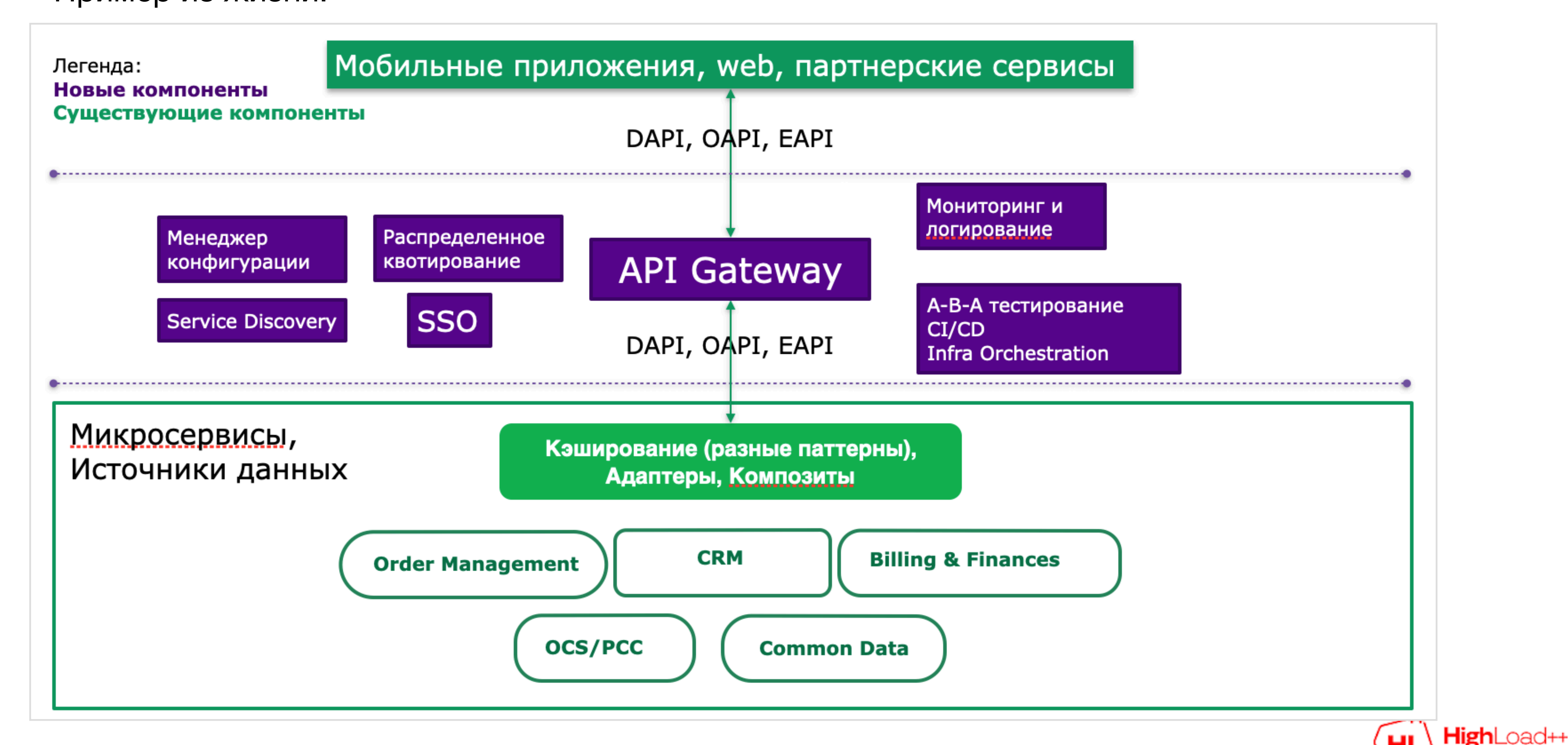
- AGW — единственная точка входа (SEP), также она располагается около приложений (Sidecar proxy).
- Ключевое отличие: приложения общаются между собой только через Sidecar proxy.
- Отчего Service Discovery, Tracing, Logging и т.п. функции AGW становятся необходимыми.





Примечание:
Зонтик, а не Service Mesh

- Когда-то давно наблюдал переходный период между Service Mesh и Микросервисами, в таком случае AGW располагается “зонтиком”.
Пример из жизни:



Service Mesh
&
Service Discovery

- Когда лучше ***не*** использовать:
  - Когда нет предпосылок, т.е. DSO как процесса, отстроенной инфраструктуры (AGW, сервис мониторинга и т.п.).
  - Как мне кажется, строить это для небольшого одного проекта смысла не имеет, это увеличит и сроки проекта, и его стоимость.



Service Mesh
и
Service Discovery

- Когда лучше использовать:
    - Когда у вас выстроена инфраструктура, есть регламенты, есть нечто похожее на DSO и как процесс, и как набор технологий
    - Есть большое кол-во проектов, которое нужно уметь контролировать системно



Добавим к выводу

- Должен использоваться, только если компания большая ***и*** имеет планы роста, оптимизаций и т.п.
- Открывает возможность монетизации API, продажи аналитики
- Может уменьшить TCO, TTM, а может и увеличить
- AGW улучшает безопасность, улучшает контроль, поддержку ряда функций
- AGW без “процессов” — сомнительная история



Спасибо! Вопросы?

Контакты: vasiliy.soshnikov@gmail.com

